A CSP-Based Rehearsal Scheduler

Monica Song

December 8, 2017

1 Introduction

Tech week in the world of dance and theater is a major headache for producers. It requires booking back-to-back rehearsals in the theater space, making the most out of the stage prior to opening night.

As tech producer for the Harvard Ballet Company, I had the responsibility of scheduling our tech week. This meant accommodating the schedules of 30+ busy college students to make sure that our tech week was as efficient and effective as possible, which was quite the undertaking.

The producer has always made the tech week schedule by looking at a spreadsheet of everyones availability and then through a method of trial-by-error, figuring out the best time slots that work for the most people. However, you never know if the final rehearsal schedule is optimal. Inevitably, there are always dancers and musicians who simply cant make all their scheduled rehearsal times.

1.1 Goals

This project aims to offer a fast and optimal solution to scheduling tech week. There are several methods that the scheduler provides:

- Depth first search with a pruned tree
- Heuristic-based search with forward-checking
- Simulated annealing

Each of these algorithms caters to a different level of difficulty. DFS is recommended for easier CSPs in which there is ’wiggle room’, i.e. one doesn’t need to relax many soft constraints. Heuristic search is recommended if one needs to find an efficient solution for medium-size problems that have 10 - 15 dancers and larger domains. Simulated annealing is ideal for creating schedules for a large number of dancers and pieces of varying priorities, which means that must experiment with different combinations of constraint violations.

If possible, one should run DFS since it is exhaustive and will deliver the optimal solution (no constraints violated) that maximizes the evaluation scores if one such exists.

1.2 Scope

The scheduler should be able to accommodate any number of dancers and pieces. In this project,I tested the project on a maximum of 18 dancers and 20 pieces, since these were the most I had in my data. However, the algorithms should be able to scale easily. Furthermore, from a realistic point of view, the scheduler will never need to accommodate more than 40 dancers, since this is upper bound of members in the Harvard Ballet Company.

2 Background and Related Work

2.1 Motivation

While creating the tech week schedule for the Harvard Ballet Company's performance of *In Passage* this past October, I knew I wanted to automate this often 2-hour long process, both to make the process easier for myself and for future producers of HBC.

2.2 Course Connections

The project draws heavily from several topics in the course, especially those described in Chapter 6 of *Artificial Intelligence: A Modern Approach*:

- **Search Tree with Pruning**: I implemented a search tree, each node representing a time slot assignment for a choreographer piece. While traversing the tree, if we found that a node contained an impossible assignment, we skipped adding the node's children to the stack of nodes to check.

- **Node and Arc Consistency**: I made sure that nodes were consistent by limiting the domain of each variable to include the intersection of values in the choreographer's and dancers' availabilities.

  As long as no node had an empty domain, the problem satisfied the requirements of single arc consistency. I reasoned that enforcing K-consistency was unnecessary because the graph representation of the constraints is a clique (in that every rehearsal slot must be a different time); thus, ensuring K-consistency would have effectively solved the problem.

- **Heuristics**: In terms of ordering the levels of my tree, I used the minimum-remaining-values heuristic to determine an order of the pieces. Additionally, I used the least constraining values heuristic in assigning values for each variable.

- **Local Search**: I used simulated annealing to solve problems for which depth first search took too long. This type of problem typically has many dancers and pieces; thus it would be reasonable if we violated several soft constraints.

2.3 Project Resources

I mainly referenced the course text *Artificial Intelligence: A Modern Approach* for my project. For defining my local search solution, I also referred to paper from the University of British Columbia about solving weighted maximum constraint satisfaction problems. The authors suggest using the

Min-Conflict Heuristic with iterated local search to solve the the NP-hard problem of constraint optimization [1].

Additionally, I used the code from Problem Set 2 *sudoku.py* to help architect the structure of the scheduler.

3 Problem Specification

Given a list of dancer's schedules and choreographer's pieces, I formulated the task of assigning times to each piece as constraint satisfaction problem.

- **Variables**: The hour-long slot of a choreographers rehearsal
- **Domains**: The times that the stage is open, typically 11am to 11pm but can vary
- **Constraints**:
  - Hard Constraints: Choreographers availability. It is necessary that a choreographers rehearsal be scheduled during a time which they are available.
  - Soft Constraints: The dancers in the choreographers piece availability. It would be most ideal if all the dancers in a choreographers piece could attend the rehearsal slot.
  - Nice-to-haves:
    - Non-Harvard dancers have all their rehearsals scheduled on the same day, within 3 hours of each other, so they dont have to travel
    - Rehearsals clustered together
    - Dinner breaks (a hour long break between 5:30 and 7:30pm)

I also specified an evaluation function to rate the solution. The evaluation functions assigns points based on the number of soft constraints met, the 'cluster factor' (i.e. if a dancer's rehearsals are close together in a day), and if other 'nice-to-haves' that are satisfied.

4 Approach

My program first checks if all variables have non-empty domains. If there are variables with empty domains, my program selects removes the "most constraining" dancer from the piece, i.e. the dancer whose availability overlaps the least with the choreographer's availability. Next, it runs the assignment algorithm. If no solution is found, the program relaxes a constraint, selecting the piece with minimum remaining values and removing the most constraining dancer.It continues to perform this process until it arrives at a solution.

I provide several algorithms to address the varying amounts of constraints and variables for each CSP.

4.1 Depth First Search with Pruning

Each node in the search tree represents a piece and an assigned slot. My algorithm for DFS keeps track of the paths to each node. It checks that the state of each node is not contained in the path of its predecessors. If it is, this means that the time assignment in the node has already been assigned to another node.

Algorithm 1 DFS with pruning

```
procedure DFS(start_node)
    while stack do vertex = stack.pop()
        if vertex.successors then
            for s in successors do
                if s not in vertex.path then
                    $stack.add(s)$
```

4.2 Heuristic Assignment

While DFS provides all the solutions possible, heuristic-based search results in a single solution assignment. It assigns the current least constraining value to the queue of variables ordered by the the number of remaining. If the algorithm finds that LCV results in a variable with an empty domain, then it replace the value of the variable with the next least constraining value.

Algorithm 2 Heuristic Assignment with Forward Checking

```
procedure HEURISTIC(pieces)
    while $order(pieces)$ do
        $MRV$ = order[0]
        $MRV.slot$ = LCV
        while $forward\_check()$ = False do
            $MRV.slot$ = next LCV
```

4.3 Simulated Annealing

Simulated annealing provides an approximate best solution under a given schedule. We select neighbors from the list of pieces with violations and assign to it the value with minimal conflict. We evaluate each assignment based its length of constraint violations.

Algorithm 3 Simulated Annealing

```
procedure SIMULATED ANNEALING(pieces)
    while $T > 0$ do
        $current\_value$ = len(violations)
        $rand\_piece$ = random.choice(violations)
        $assign\_to\_neighbor(rand\_piece)$
        $new\_value = len(violations)$
        if $new\_value < current\_value$ then
            accept $new\_value$
        else
            accept $new\_value$ with probability p
```

4.4 Assumptions

While building the scheduler, there were several assumptions and approximations I made that were specific to my own experience as a member of the Harvard Ballet Company. First of all, my evaluation function prefers factors such as the allowance of dinner breaks, short campus travel time for non-Harvard college performers, and larger chunks of rehearsal time. These factors are not objectively great and may not actually be preferred by dancers. For instance, is it preferable that a dancer's rehearsals be clustered together or is it better that they be spaced out farther from each other?

When the program needed to relax a constraint, there were several options for which piece to select:

- The piece with the most dancers
- The piece that is the most "performance-ready" (i.e. it is not necessary that all dancers attend the rehearsal)
- The piece with the smallest domain

In terms of deciding which constraints to relax for a given piece, again there were several options for which dancer to remove:

- The busiest dancer (i.e. lowest availability)
- The dancer in the most pieces
- The most constraining dancer

My algorithm selected the last option for both of the choices above to determine which constraint to relax, as these typically have the lowest weights for constraint satisfaction problems. However, this may not result in the schedule that truly maximizes the stage time prior to opening night.

5 Experiments

My data came directly from the spreadsheets in which dancers entered their availabilities and the actual tech week schedule created that semester. I ran each of the three algorithms on each of the data sets. To measure the time of each algorithm, I ran the program, adding the word `time` before the command in the terminal. I then took an average of 10 trials. Finally, I compared the results of my program with the actual tech week schedule of the performance.

5.1 Data

I used 3 data sets, each one containing the availabilities of the choreographers and non-Harvard dancers as well as the tech week schedule that was created. *Note: links can only be viewed by Harvard College users.*

- *In Passage*, Fall 2017: dancer availabilities, tech week schedule
  - This data set has 17 dancers and 7 pieces.

  - It was impossible to find a solution for this data set that did not violate a hard constraint since two variables had empty domains to begin with

- *CityScapes*, Spring 2017: dancer availabilities, tech week schedule
  - This data set has 18 dancers and 10 pieces.
  - This data set was unique in that it the number of values in the domain was equal to the number of variables.

- *Oz*, Fall 2016: dancer availabilities, tech week schedule
  - This data set has 14 dancers and 14 pieces.
  - This data set had the most variables since each choreographer had two rehearsals: lighting and blocking.

The objects that represent the dancers and pieces for each show can be found in *inPassage.py*, *CityScapes.py*, and *Oz.py*.

5.2 Results

I tested each data set using the three algorithms. I evaluated each one and compared it to the tech week schedule that was created the ”traditional” way, i.e. using pen and paper. For the randomized simulated annealing algorithm, I recorded the best run out of 20 trials. The evaluation scores and number of violations represented as a tuple are in Table 1 and evaluation times in Table 2. *Note: I did not test DFS on the Oz data set due the size of tree which had over 10^{11} nodes.*

Algorithm	DFS	Heuristic	SA	Traditional
In Passage	(42, 2)	(42, 2)	(42, 2)	(42.5, 6)
CityScapes	(33.3, 0)	(22.8, 0)	(32.0, 0)	(32.3, 2)
Oz	–	(16.7, 1)	(11.5, 2)	(24.5, 12)

Table 1: Evaluation Scores, Number of Violations

Algorithm	DFS	Heuristic	SA	Traditional
In Passage	0.1sec	0.037sec	0.045sec	1hr
CityScapes	0.32sec	0.039sec	0.045sec	1hr
Oz	–	0.039s	0.052sec	1hr

Table 2: Evaluation Times

6 Discussion

Overall, the process of implementing the scheduler was fairly smooth. Two issues I faced were deciding how to define the neighbors for simulated annealing and how to weight constraints satisfied for the evaluation function. From these results, one can see that the scheduler has achieved its goal of efficiently creating an optimal schedule when possible. It outperforms the traditional method consistently in terms of violations. However, in terms of maximizing the evaluation function, the algorithm could use some tweaking in order to capture the full interests of the ballet company.

6.1 Summary

In general, the heuristic algorithm was the best. It had the fastest runtime and delivered the best results the majority of the time. The heuristic algorithm was similar to the depth first search algorithm. However, it only traveled down the length of the tree once and assigned values to nodes based on the Least Constraining Values heuristic. By forward checking with each assignment, this algorithm only had to backtrack by one level. This algorithm is an example of the power of heuristics when approximating the optimal solution.

Simulated annealing also performed extremely well. It performed just as well as DFS and the heuristic algorithm for the *InPassage* data set. It even beat the heuristic algorithm around 10% of the time using the *CityScapes* data set. The parameters I used to decide the schedule for simulated annealing were fairly conservative, resulting in around 90 iterations of the SA algorithm.

And lastly, depth first search was fairly powerful and fractions of a second slower than the other two algorithms. It delivered the optimal solution when possible, as evidenced by the *CityScapes* data set. However, for larger problems when one needs to relax several constraints and thus increase the number of possible values for each variable, DFS would take over 10 minutes to run, making it un-ideal for such a task.

If the number of nodes in the search tree is below 10 billion, I would recommend using DFS for determining the tech week schedule. Otherwise, I believe simulated annealing is the way to go. Because it is a randomized algorithm, one can general different solution with each run, so one can simply pick the one that best satisfies their needs.

6.2 Lessons

I learned a lot in making this project. One of the most valuable lessons was in defining the evaluation function, which needs to capture human elements such as the need for dinner breaks and the need to account for transportation time. My program does not have the best evaluation function, since the traditional method is still able to score the highest. Thus, if given more human-made tech week schedules, one could train a model to determine the best evaluation function that captures the full spectrum of latent factors and interaction terms.

6.3 Future Improvements

Given more time, I would have liked to implement these points of functionality:

- More optimal constraint relaxation: Each time the algorithm arrives at an invalid assignment, it relaxes a constraint such that the number of constraints increases by one each iteration. As opposed to relaxing an additional constraint after each invalid assignment, I would like to cycle through all the different possible constraint relaxations before adding one more.

- Allowing the user to select which constraint to relax: Because the user likely knows which constraints are less necessary, he or she would better determine what constraints to relax to produce the best tech week schedule. We can see that this is the case for the *InPassage* data set, in which the traditional methods had a slightly higher evaluation score.

A System Description

A.1 Running the System

All the code for the scheduler is available on Github at this link:
https://github.com/monicadsong/CS182-Final-Project.

To run the code, use:

```
python solver.py <show> <algorithm>
```

where `<show>` can be replaced with `InPassage`, `CityScapes`, or `Oz`, and `<algorithm>` can be replaced with `DFS`, `heuristic`, or `random`.

An example command is: `python solver.py InPassage heuristic`

To evaluate the actual tech week schedule from the performance, use:

```
python eval_solution.py <show>
```

A.2 Understanding the Output

The program prints out the rehearsal assignments on stdout. First, it prints out the assigned rehearsal time for each choreographer's piece. Then, it prints out the rehearsal schedules for each dancer. Lastly, it prints out the constraint violations and evaluation score.

References

[1] Diana Kapsa and Jacek Kisynski. Solving the weighted maximum constraint satisfaction problem using dynamic and iterated local search. *University of British Columbia*, 2003.